Statistical Techniques in Robotics (16-831, S20) Lecture #17 (Wednesday, April 7)

Model-Free Value Prediction

Lecturer: Kris Kitani *Scribes: Tanay Sharma, Paritosh Mittal*

1 Review

In the last lecture, on the topic of Value Based Reinforcement Learning (RL) algorithms, we learned Value functions and their recurrent relationship was stated using Bellman Equations. The Value Based RL paradigm can be understood by Fig 1. Since we explicitly estimate value functions to estimate the best policy, these methods are termed as Value Based. There are mainly two types of algorithms under this (1) Model-Based algorithms (2) Model-Free Algorithms. We studied about Model Based algorithms. It means that the model is fully specified and we have the transition dynamics and reward function already. Under model based methods we learned two algorithms (1) Policy Iteration (2) Value Iteration.

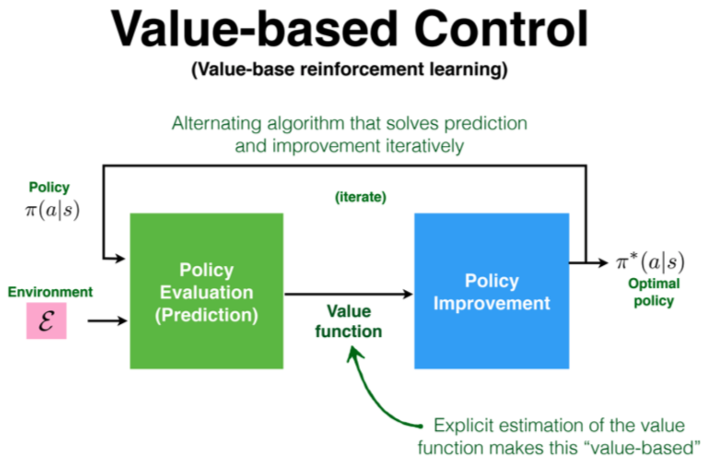
Value-based Control
(Value-base reinforcement learning)
Alternating algorithm that solves prediction and improvement iteratively
Policy
$\pi(a|s)$
(iterate)
Environment
$\mathcal{E}$
Policy Evaluation (Prediction)
Value function
Policy Improvement
$\pi^*(a|s)$
Optimal policy
Explicit estimation of the value function makes this "value-based"

Figure 1: Caption

1.1 Policy Iteration

To understand this algorithm we recall the Bellman Equations which are given as :

$$\begin{aligned} V^{\pi}(s) &= \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a)\left[r(s', a, s) + \gamma V^{\pi}(s')\right] \\ Q^{\pi}(s, a) &= \sum_{s'} p(s' \mid s, a)\left\{r(s', a, s) + \sum_{a'} \pi(a' \mid s') Q^{\pi}(s', a')\right\} \end{aligned} \tag{1}$$

Policy iteration builds on this recursive algorithm that we know needs to be satisfied. As can be seen in Fig 1, there are two steps in value based control algorithms (1) Policy Evaluation (Prediction) (2) Policy Improvement.
The algorithm for policy iteration is shown below :

Algorithm 1 POLICY_ITERATION $(\pi, r(s), p(s' \mid s, a), \gamma)$

```
1:  $\pi \leftarrow \text{rand}(\mathcal{A})$
2:  $V \leftarrow \text{rand}(\mathbb{R})$
3:  $V' \leftarrow \text{rand}(\mathbb{R})$
4:  while $\max_s |V(s) - V'(s)| \geq \epsilon$ do
5:          $V' \leftarrow V$
6:      for $s \in \mathcal{S}$ do
7:          $Q(s,a) = r(s) + \gamma \sum_{s'} p(s' \mid s, a) V'(s') \quad \forall a$
8:          $V(s) = \sum_a \pi(a \mid s) Q(s,a)$
9:      end for
10: end while
11: for $s \in \mathcal{S}$ do
12:     $\pi'(s) = \arg\max_a Q(s,a)$
13: end for
14: if $\pi' = \pi$ then
15:     return $\pi$
16: end if
    Go to line 1
```

According to the algorithm, policy evaluation is done from line 6 to 8. We calculate the new estimate of our value function given a policy using the current estimate and update the current estimate with it. We continue this process until the change in consecutive value is below a certain threshold (convergence). Once we have the optimum values of our value function for given policy we go to the policy improvement stage. We know that when the policy is optimal it will be equal to max(Q Value function) (line 11-13). Now the whole process is repeated until our policy converges to an optimal value

1.2 Value iteration

Value iteration is a modification of Policy iteration to make the algorithm faster. As policy was updated after value function converges in policy iteration, it takes a lot of time for policy to converge. Value iteration does the policy improvement in the while loop after policy evaluation at each step (after line 8 above). This leads to faster convergence

2 Model Free Algorithms

Now we will look into Model Free Algorithms that are another set of algorithms for Value based RL. Unlike model based algorithms, we do no have direct access to the model. Hence, we do not have the transition dynamics and reward function. The only way to calculate the value function to find the optimal policy is by interacting with environment. We can generate many samples by this interaction and samples that we receive can be in the form :

$$\{s^{'}, r, s, a\}_1^N \tag{2}$$

where $s^{'}$ = new state, r = reward, s = current state, a = action taken, N = number of samples These types of algorithms are useful in many real-life scenarios where possible states can be huge. In such scenarios, calculating model parameters like transition dynamics and reward function may be tedious, prone to errors or not possible. There are two methods mainly to calculate the value function from this sampled interaction with the environment.

1. **System Identification ("Model-based RL"):** In this method we collect many samples from the environment by interaction and convert the model free problem to model based problem by using the samples as data/knowledge about the environment. Once we parameterize the transition dynamics and reward function, we can use the algorithms as described above.

2. **Model-free Prediction:** Under these methods , we directly estimate the value function while interacting with the environment. We will look into two categories of such algorithms :

   (a) Monte Carlo

   (b) Temporal Differencing

2.1 Monte Carlo Prediction

Value of a state is the expected (mean) return under the MDP (policy, dynamics, initial state distribution

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi,P}\left[\sum_{t=0}^{T} \gamma^t r_t \mid s_0 = s\right] \\ &= \mathbb{E}_{\pi,P}\left[G \mid s_0 = s\right] \end{aligned}$$

We cannot calculate this quantity directly in model-free method as we do not have access to transition dynamics. Monte Carlo method estimates the value function by interacting with the environment many times and using the average return value as the value function at that step.

$$V(s) = \frac{1}{N}\sum_{i=1}^{N} G_i(s) \tag{3}$$

where $G_i(s)$ is the total reward at a time step (iteration) i and N is the total number of interactions with the environment starting from state s. There are different variants of Monte Carlo estimation methods :

1. FirstVisit-MC-Prediction

2. FirstVisit-Incremental-MC-Prediction

3. FirstVisit-Dynamic-MC-Prediction

4. EveryVisit-MC-Prediction

Let us discuss each algorithm independently,

First Visit MC Prediction

Algorithm 2 FIRSTVISIT-MC-PREDICTION (π)

```
1: for $e = 1, \dots, E$ do
2:   $\left\{s^{(t)}, a^{(t)}, r^{(t)}\right\}_{t=0}^{T} \sim \mathcal{E} \mid \pi$
3:   for $t = 0, \dots, T$ do
4:     if first visit to $s^{(t)}$ in episode $e$ then
5:       $G\left(s^{(t)}\right) \leftarrow G\left(s^{(t)}\right) + \sum_{i=t}^{T} r^{(i)}$
6:       $N\left(s^{(t)}\right) \leftarrow N\left(s^{(t)}\right) + 1$
7:     end if
8:   end for
9: end for
10: return $V(s) \leftarrow G(s)/N(s) \forall s$
```

Key points for First Visit Monte Carlo Prediction:

- Line 2: of algo. 2 indicate that for each episode e we sample T experiences from the environment $\mathcal{E}$ using the policy π.
- The algo. only updates $G(s^{(t)})$ once per episode when the state $s^{(t)}$ is first visited
- The update to $G(s^{(t)})$ is the cumulative rewards received from time step t to T.
- A counter is maintained to keep track of the number of episodes that reach state $s^{(t)}$
- Final value function is updated at the very end of the algorithm (Line 10:).

Another variant of Monte Carlo algorithm builds on the concept of incremental updates.

First Visit Incremental MC Prediction

Algorithm 3 FIRSTVISIT-Incremental-MC-PREDICTION (π)

```
1: for $e = 1, \dots, E$ do
2:   $\left\{s^{(t)}, a^{(t)}, r^{(t)}\right\}_{t=0}^{T} \sim \mathcal{E} \mid \pi$
3:   for $t = 0, \dots, T$ do
4:     if first visit to $s^{(t)}$ in episode $e$ then
5:       $G\left(s^{(t)}\right) = \sum_{i=t}^{T} r^{(i)}$
6:       $N\left(s^{(t)}\right) \leftarrow N\left(s^{(t)}\right) + 1$
7:       $V(s) \leftarrow V(s) + \frac{1}{N(s^{(t)})}(G(s^{(t)}) - V(s^{(t)}))$
8:     end if
9:   end for
10: end for
11: return $V(s)$
```

Key points for First Visit Incremental Monte Carlo Prediction:

- Line 7: of algo. 3 contains incremental updates in the Value function $V(s)$
- Incremental updates are done using a moving mean based estimate of $V(s)$

The benefit of incremental updates is to have a intermediate estimate. This is useful as it enables us to further modify the algorithm to include non-stationary environment model (What if the environment also changes with time, then it is sensible to 'forget' or give less weight to old episodes). This brings us to a third variant of Monte Carlo Algorithms.

First Visit Dynamic Monte Carlo Prediction

Algorithm 4 FIRSTVISIT-Dynamic-MC-PREDICTION (π, α)

```
 1: for $e = 1, \ldots, E$ do
 2:   $\{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim \mathcal{E} \mid \pi$
 3:   for $t = 0, \ldots, T$ do
 4:     if first visit to $s^{(t)}$ in episode $e$ then
 5:       $G\left(s^{(t)}\right) = \sum_{i=t}^{T} r^{(i)}$
 6:       $V(s) \leftarrow V(s) + \alpha(G(s^{(t)}) - V(s^{(t)}))$
 7:     end if
 8:   end for
 9: end for
10: return $V(s)$
```

Key points for First Visit Dynamic Monte Carlo Prediction:

- An α parameter $(0, 1)$ is used to control the impact of recent episodes.
- The hyper-parameter adds more variance to the Monte Carlo algorithm.

Algorithm 5 EVERYVISIT-MC-PREDICTION (π, α)

```
1: for $e = 1, \ldots, E$ do
2:   $\{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim \mathcal{E} \mid \pi$
3:   for $t = 0, \ldots, T$ do
4:     $G\left(s^{(t)}\right) = \sum_{i=t}^{T} r^{(i)}$
5:     $V(s) \leftarrow V(s) + \alpha(G(s^{(t)}) - V(s^{(t)}))$
6:   end for
7: end for
8: return $V(s)$
```

Key points for Every Visit Monte Carlo Prediction:

- Algorithm makes update for every time step t instead of only once per episode
- Algorithm does not need to keep track of visits

Properties of Monte Carlo Algorithms

- Monte Carlo algorithms are model free algorithms wherein the algorithm does not know MDP transitions
- The agent learns from sampled experiences. However, it can only work for finite horizon or episodic problems
- Monte Carlo algorithms update $G(s^{(t)})$ as sum over all rewards of sub-trajectory. For very large finite horizons (T), this compounding effect results in very large variance.
- Monte Carlo algorithms use many samples (full returns) within an episode to estimate value function (they do not use bootstrapping).

Temporal Difference Prediction

We first define a **Bootstrapping** method as a method that computes an estimate based on another estimate. In Reinforcement Learning paradigm, we aim to estimate a Value function ($V(s)$), however bootstrapped methods estimate $V(s)$ using some other estimates. Monte Carlo algorithms are not bootstrapped because they estimate $V(s)$ using full returns. Temporal Difference on the other end is a bootstrapped method.

Temporal Difference (TD) is a model free method of estimating the Value function $V(s)$. In contrast with Monte Carlo algorithms which use all returns from an episode, TD estimates $V(s)$ from incomplete episode by using the current reward and previous estimate of $V(s)$. TD is generally regarded as a combination of Monte Carlo algorithms and Dynamic Programming. We know the update of $V(s)$ as discovered in previous section is

$$V(s) \leftarrow V(s) + \alpha(G_t - V(s))$$

In model-free approach, if the environment returns $\langle r_t, s_{t+1}\rangle$ for current state - action pair $\langle s_t, a_t\rangle$, then G_t can be estimated as the sum of current reward (r_t) and estimated value of next state ($V(s_{t+1})$):

$$G_t = r_t + \gamma V(s_{t+1})$$

The above expression is often regarded as Temporal Difference target (or TD target). Since at any stage we have access to only one-step look ahead, this TD target is known as 1-step TD target. The goal is then to estimate this target. Replacing it in the above expression gives us

$$V(s) \leftarrow V(s) + \alpha(r_t + \gamma V(s_{t+1}) - V(s))$$

Note that term $r_t + \gamma V(s_{t+1}) - V(s)$ is the difference between our TD target and the current estimate of Value function $V(s)$. Hence the expression $r_t + \gamma V(s_{t+1}) - V(s)$ is often termed as TD error.

$$\begin{aligned} \textbf{TD Target :}\ & r_t + \gamma V(s_{t+1}) \\ \textbf{TD Error :}\ & r_t + \gamma V(s_{t+1}) - V(s) \end{aligned}$$

Algorithm 6 1-STEP-TD-Prediction (π, α)

```
1: for $e = 1, \ldots, E$ do
2:     $\left\{s^{(t)}, a^{(t)}, r^{(t)}\right\}_{t=0}^{T} \sim \mathcal{E} \mid \pi$
3:     for $t = 0, \ldots, T-1$ do
4:         $G^{(t)}\left(1\right) \leftarrow r^{(t)} + \gamma V(s^{(t+1)})$
5:         $V(s^{(t)}) \leftarrow V(s^{(t)}) + \alpha\left[G^{(t)}(1) - V(s^{(t)})\right]$
6:     end for
7: end for
8: return $V(s)$
```

The algorithm 6 explains the implementation of 1 step Temporal Difference based Value function Estimation. *Notation:* Here $G^{(t)}\left(1\right)$ indicates the total reward (G) at time t using 1 step TD. Similarly $G^{(t)}\left(2\right), G^{(t)}\left(3\right), G^{(t)}\left(4\right), \cdots$ represent total reward using $2, 3, 4 \cdots$ step TD respectively. Mathematically, they can be defined as

$$\begin{aligned}
G^{(t)}\left(1\right) =& r^{(t)} + \gamma V(s^{(t+1)}) \\
G^{(t)}\left(2\right) =& r^{(t)} + \gamma r^{(t+1)} + \gamma^{2} V(s^{(t+2)}) \\
& \vdots \\
\textbf{TD N step} & \\
G^{(t)}\left(N\right) =& r^{(t)} + \gamma r^{(t+1)} + \cdots + \gamma^{(N-1)} r^{(N-1)} + \gamma^{N} V(s^{(t+N)}) \\
& \vdots \\
\textbf{Monte Carlo} & \\
G^{(t)}\left(\infty\right) =& r^{(t)} + \gamma r^{(t+1)} + \cdots + \gamma^{T} r^{T}
\end{aligned}$$

Hence the algorithm for N-step TD based prediction will be

Algorithm 7 N-STEP-TD-Prediction (π, α)

```
1: for $e = 1, \ldots, E$ do
2:     $\left\{s^{(t)}, a^{(t)}, r^{(t)}\right\}_{t=0}^{T} \sim \mathcal{E} \mid \pi$
3:     for $t = 0, \ldots, T-1$ do
4:         $G^{(t)}\left(N\right) \leftarrow \sum_{i=t}^{t+N-1} \gamma^{i-t} r^{(i)} + \gamma^{N} V(s^{(t+N)})$
5:         $V(s^{(t)}) \leftarrow V(s^{(t)}) + \alpha\left[G^{(t)}(N) - V(s^{(t)})\right]$
6:     end for
7: end for
8: return $V(s)$
```

The Computation of algorithm 7 for N-step TD can be greatly improved by iterating backwards and caching the sum for rewards. Also, the TD algorithms can perform better than Monte Carlo algorithms and can work with infinite horizon mainly because it uses bootstrapping.

3 Appendix

3.1 Markov Chain Monte Carlo Algorithms

Monte Carlo is a technique for randomly sampling a probability distribution and approximating a desired quantity[4]. A Markov chain is a Markov process with discrete time and discrete state space. Markov chain is a discrete sequence of states, each drawn from a discrete state space (finite or not), and that follows the Markov property[2]. Markov property states that the probability distribution of the next state only depends on the current state and not on the history.

MCMC is essentially Monte Carlo prediction using Markov chains. Monte Carlo method draws samples from the the required distribution, and then calculate sample averages after multiple iterations to approximate expectations. Markov chain Monte Carlo draws these samples by constructing a Markov chain of some horizon (length of Markov Chain) for a multiple iterations. There are different algorithms that change in the way they form the Markov Chains for Monte Carlo estimation for the desired quantity [4]

3.2 Gibbs Sampling Algorithm

Gibbs sampling algorithm is an MCMC algorithm that is useful when we want to estimate the multivariate posterior probability distribution. This works when we can sample from conditional probability distribution of random variables easily but calculating a joint posterior distribution is difficult. Let us assume there are two random distributions X and Y. The Gibbs sampling then works as follows

```
initialize $Y^0, X^0$
for j = $1, 2, 3, \ldots$ do
    sample $X^j \sim p\left(X \mid Y^{j-1}\right)$
    sample $Y^j \sim p\left(Y \mid X^j\right)$          (4)
end for
```

Basically, we start with some initial values of random variables and in subsequent iterations we sample a new value of one random variable conditioned on the previous value of other random variable. This process is repeated until convergence which approximates the sampled values as they were drawn from real joint posterior distribution [3]

3.3 Metropolis-Hastings Algorithm [1]

This algorithm directly estimates the joint posterior probability and does not need the conditional probability distribution of each random variable as was required in Gibbs Sampling. It uses an independent proposal distribution and decides to accept the proposal at each step. The algorithm works like this (1) Generate a candidate sample x_{cand} from the proposal distribution $q(x(i)|x(i-1))$ (2)Use the acceptance function $\alpha(x_{cand}|x(i-1))$ and compute the acceptance probability based upon the proposal distribution and the full joint density $\pi(.)$ (3) Accept the candidate sample with probability α, the acceptance probability, or reject it with probability $1-\alpha$

Algorithm 8 Metropolis-Hastings algorithm

```
1:  Initialize $x^{(0)} \sim q(x)$
2:  for iteration $i = 1, 2, \ldots$ do
3:     Propose: $x^{\text{cand}} \sim q\left(x^{(i)} \mid x^{(i-1)}\right)$
4:     Acceptance Probability:
5:         $\alpha\left(x^{cand} \mid x^{(i-1)}\right) = \min\left\{1, \frac{q\left(x^{(i-1)} \mid x^{cand}\right)\pi\left(x^{cand}\right)}{q\left(x^{cand} \mid x^{(i-1)}\right)\pi\left(x^{(i-1)}\right)}\right\}$
6:     $u \sim$ Uniform $(u; 0, 1)$
7:      if $u < \alpha$ then
8:      Accept the proposal: $x^{(i)} \leftarrow x^{\text{cand}}$
9:      else
10:        Reject the proposal: $x^{(i)} \leftarrow x^{(i-1)}$
11:     end if
12: end for
```

References

[1] Ilker Yildirim *Bayesian Inference: Metropolis-Hastings Sampling*. Department of Brain and Cognitive Sciences University of Rochester
`http://www.mit.edu/ ilkery/papers/MetropolisHastingsSampling.pdf`

[2] *Joseph Rocca* [*Introduction to Markov chains*].
`https://towardsdatascience.com/brief-introduction-to-markov-chains-2c8cab9c98ab`

[3] Cory Maklin, Gibbs Sampling
`https://towardsdatascience.com/gibbs-sampling-8e4844560ae5`

[4] Jason Brownie, A Gentle Introduction to Markov Chain Monte Carlo for Probability
`https://machinelearningmastery.com/markov-chain-monte-carlo-for-probability/`